

# GitOps

Introduction

Ravindu Nirmal Fernando | SLIIT | February 2025

# GitOps Principles

v0.1.0

## 1 The principle of declarative desired state

A system managed by GitOps must have its Desired State expressed declaratively as data in a format writable and readable by both humans and machines.

## 2 The principle of immutable desired state versions

Desired State is stored in a way that supports versioning, immutability of versions, and retains a complete version history.

## 3 The principle of continuous state reconciliation

Software agents continuously, and automatically, compare a system's Actual State to its Desired State. If the actual and desired states differ for any reason, automated actions to reconcile them are initiated.

## 4 The principle of operations through declaration

The only mechanism through which the system is intentionally operated on is through these principles.

# GitOps in K8s

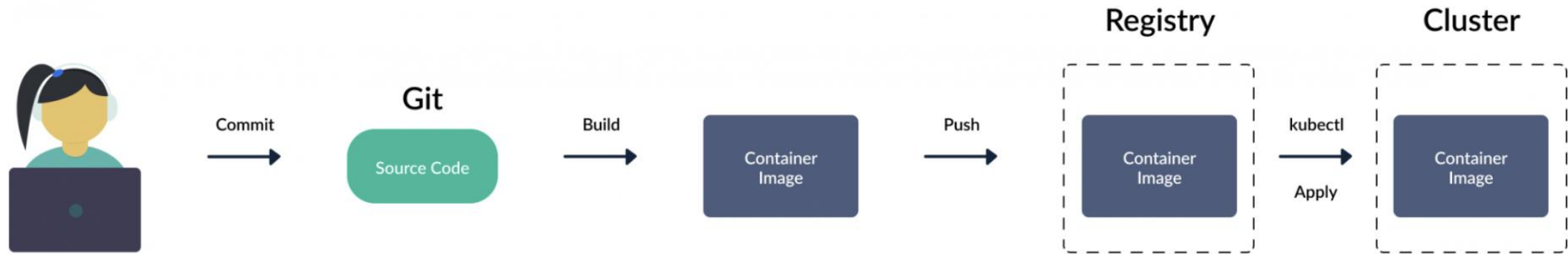
In the case of Kubernetes, GitOps deployments happen in the following manner:

A GitOps agent is deployed on the cluster.

- The GitOps agent is monitoring one or more Git repositories that define applications and contain Kubernetes manifests (or Helm charts or Kustomize files).
- Once a Git commit happens the GitOps agent is instructing the cluster to reach the same state as what is described in Git.
- Developers, operators, and other stakeholders perform all changes via Git operations and never directly touch the cluster (or perform manual kubectl commands).

# Traditional deployment without GitOps:

- 1 - A developer commits source code for the application.
- 2 - A CI system builds the application and may also perform additional actions such as unit tests, security scans, static checks, etc.
- 3 - The container image is stored in a Container registry.
- 4 - The CI platform (or other external system) with direct access to the Kubernetes cluster creates a deployment using a variation of the “kubectl apply” command.
- 5 - The application is deployed on the cluster.



- The cluster state is manually decided by `kubectl` commands or other API access.
- The platform that deploys to the cluster is having full access to the Kubernetes cluster from an external point.

# Modifying the process with GitOps

The first steps are the same.

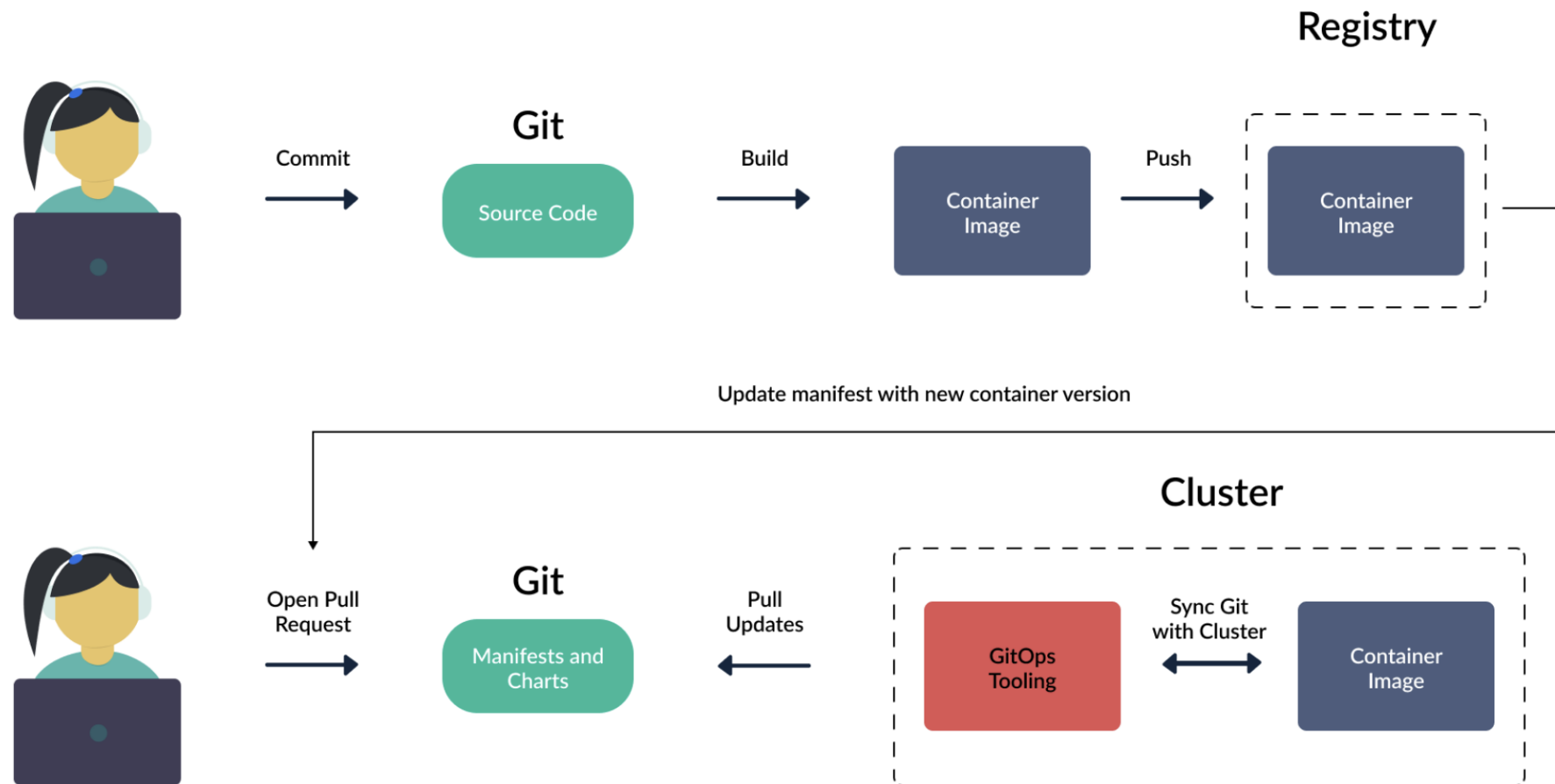
1 - A developer commits source code for the application and the CI system creates a container image that is pushed to a registry.

2 - Nobody has direct access to the Kubernetes cluster.

3 - There is a second Git repository that has all manifests that define the application.

4 - Another human or an automated system changes the manifests in this second Git repository.

5 - A GitOps controller that is running inside the cluster is monitoring the Git repository and as soon as a change is made, it changes the cluster state to match what is described in Git.



The key points here are:

- The state of the cluster is always described in Git. Git holds everything for the application and not just the source code.
- There is no external deployment/CI system with full access to the cluster. The cluster itself is pulling changes and deployment information.
- The GitOps controller is running in a constant loop and always matches the Git state with the cluster state.